

DOCUMENT RESUME

ED 071 683

LI 004 078

AUTHOR Foulk, Clinton R.; Juelich, Otto C.
TITLE Smooth Programs and Languages.
INSTITUTION Ohio State Univ., Columbus. Computer and Information
Science Research Center..
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO OSU-CISRC-TR-72-13
PUB DATE Nov 72
NOTE 19p.; (10 References)

EDRS PRICE MF-\$0.65 HC-\$3.29
DESCRIPTORS *Computer Programs; *Computer Science; Flow Charts;
*Information Science; *Programing Languages

ABSTRACT

A smooth program is defined to be one which is "go to"-free in the sense that it can be represented by a flowchart consisting only of concatenation, alternation, and iteration elements. Three methods of eliminating the "go to" statement from a program have been proposed: (1) the introduction of additional Boolean variables or the equivalent recomputation of certain quantities in the program, (2) the use of recursive procedure calls, and (3) replacement of the "go to" statement by a restricted form of the "go to" such as the "exit" or "leave" statement. We show that only the first of these is capable of transforming a non-smooth program into a smooth one, since strict application of the recursive procedure method requires the use of a so-called "null procedure" which is in fact also a restricted form of the "go to." (Author)

(OSU-CISRC-TR-72-13)

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
OFFICE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRO-
DUCED EXACTLY AS RECEIVED FROM
THE PERSON OR ORGANIZATION ORIG-
INATING IT. POINTS OF VIEW OR OPIN-
IONS STATED DO NOT NECESSARILY
REPRESENT OFFICIAL OFFICE OF EDU-
CATION POSITION OR POLICY.

SMOOTH PROGRAMS AND LANGUAGES

by

Clinton R. Foulk and Otto C. Juelich

Work performed under

Grant No. 534.1, National Science Foundation

Computer and Information Science Research Center

The Ohio State University

Columbus, Ohio 43210

November 1972

ED 071683

I 004 078

ABSTRACT

In this paper a smooth program is defined to be one which is go to-free in the sense that it can be represented by a flowchart consisting only of concatenation, alternation, and iteration elements. Three methods of eliminating the go to statement from a program have been proposed: 1) the introduction of additional Boolean variables or the equivalent recomputation of certain quantities in the program, 2) the use of recursive procedure calls, and 3) replacement of the go to statement by a restricted form of the go to such as the exit or leave statement. We show that only the first of these is capable of transforming a non-smooth program into a smooth one, since strict application of the recursive procedure method requires the use of a so-called "null procedure" which is in fact also a restricted form of the go to.

PREFACE

This report is the result of research supported in part by NSF Grant Number GN 534.1 from the Office of Science Information Service, National Science Foundation to the Computer and Information Science Research Center, The Ohio State University.

The Computer and Information Science Research Center of The Ohio State University is an interdisciplinary research organization which consists of the staff, graduate students, and faculty of many University departments and laboratories. This report is based on research accomplished in the Department of Computer and Information Science.

The research was administered and monitored by The Ohio State University Research Foundation.

TABLE OF CONTENTS

	<u>Page</u>
Abstract	ii
Preface	iii
I. INTRODUCTION	1
II. DEFINITIONS	3
III. THE RECURSIVE PROCEDURE METHOD	5
IV. THE <u>exit</u> STATEMENT	12
V. CONCLUSIONS	14
References	16

I. Introduction

In [1] Böhm and Jacopini consider the class of programs that can be represented by flow charts consisting of connections of computational and decision nodes. They assert that there exist flow charts which cannot be transformed directly into equivalent flowcharts consisting only of the flowchart elements π , Δ , and Ω , representing concatenation, alternation, and iteration respectively. If, however, additional Boolean variables may be introduced into the program, then any flowchart may be transformed into an equivalent flowchart consisting only of π , Δ , and Ω flowchart elements. In [2], Cooper showed that the Boolean variables can be introduced in such a way as to simulate a location counter, yielding a trivial transformation of any flowchart into π , Δ , and a single Ω element. Bruno and Steiglitz [3] call a flowchart in π , Δ , Ω form a D-chart (after Dijkstra [4]) and give a formal proof of the existence of a flowchart which is not a D-chart, but which can be transformed into a D-chart by the addition of Boolean variables or by recomputation of certain quantities in the program. We shall call a program representable by a D-chart a smooth program and a language capable of expressing only smooth programs a smooth language.

Knuth and Floyd [5] indicate the existence of two additional methods of eliminating go to statements from a program, namely the use of recursive procedure calls and the introduction of a restricted form of the go to such as the exit statement, which causes a premature exit from an iteration. They seem to suggest that the procedure call method is capable of transforming a non-smooth program into a smooth one. In fact, however, the recursive procedure method requires the introduction of a so-called "null procedure"

which is different from all other procedures, including the go to procedure, in that it does not return to its point of invocation, but rather jumps to the end of the program. In ALGOL 60 [6] this null procedure can only be written as a procedure containing a go to statement. Therefore, we conclude that the null procedure is actually a restricted form of the go to and that it is specifically the go to statement, not the procedure call mechanism, which makes ALGOL 60 a non-smooth language.

II. Definitions

We begin by considering flowcharts which consist of π (concatenation) and Δ (alternation) only. Such flowcharts are series-parallel networks. They can arise from a block structured language such as ALGOL 60 by restricting the admissible statement types to computational and decision statements. Procedure invocation may also be admitted, provided the depth of recursion is bounded.

Iteration statements may be mapped in terms of recursive procedure calls directly. For instance in pseudo-ALGOL 60

for c do L:S;

can be rewritten

procedure L; if b then begin S;j;L end;

i; L;

where i, j, and b are the initialize, increment, and test components of c respectively. Thus programs containing iteration statements can be decomposed into π and Δ without introduction of additional Boolean variables provided the iterative loops are rewritten as recursive procedure calls. Alternatively if Bohm and Jacopini's Ω (iteration) flowchart element is also allowed it is possible to flowchart iterative statements directly. We now make the following definitions:

Definition 1. A smooth program is one which can be decomposed into the flowchart elements π (concatenation), Δ (alternation), and Ω (iteration) without the introduction of additional Boolean variables. More briefly a smooth program is one whose flowchart is a D-chart.

The concept may be related to work in the area of compiler design if we introduce the interval concept as defined by Cocke [7] and Allen [8]. They

define an interval as a maximal single entry subgraph of a flowchart, such that all cyclic paths pass through the entry node. This leads to the following recursive definition:

Definition 2. A single computational or decision node is a smooth interval. A two-terminal network of smooth intervals is a smooth interval if it satisfies the following restrictions: 1) the cyclic portion must be a series-parallel network leading from the entry node back to the entry node and 2) the acyclic portion must be a series-parallel network.

Definition 1'. A smooth program is a smooth interval.

Definition 3. A smooth language is a programming language which can express only smooth programs.

i-

III. The Recursive Procedure Method

Since Bruno and Steiglitz have shown that not all flowcharts are D-charts, the class of smooth programs must be a proper subclass of the class of all programs. Moreover, as we have noted, ALGOL 60 deprived of its go to statement is a smooth language, and we should therefore be able to conclude that not all ALGOL 60 programs can be written without the use of the go to statement. This conclusion is apparently contradicted by Knuth and Floyd when they indicate that go to statements can be replaced in ALGOL 60 by procedure calls and, in fact, that "'go to' is in some sense a special case of the procedure calling mechanism." As we shall see, this apparent contradiction arises because 1) they fail to point out that the null procedure is actually a restricted form of the go to, and 2) they do not distinguish carefully between programs which are go to-free because they are smooth and those which are go to-free because they are non-smooth but happen to employ some restricted form of the go to.

In replacing go to statements by procedure calls, Knuth and Floyd begin by labelling each statement of the program. Then they replace each

L:S;

by

procedure L; begin S; L' end

where L' is the successor of L. In the case of the go to statement they replace

L: go to L';

by

procedure L; L';

The composite statements of ALGOL 60 are not mentioned in this context by Knuth and Floyd. It does no violence to their analysis to specify that

1) L: if b then LT: ST else LF: SF;

be replaced by

procedure L; begin if b then LT else LF; L' end

that

2) L: for c do LG: SG;

be replaced by

procedure L; begin for c do LG; L' end

that

3) L: begin [declarations;] LG:SG; ...end

be replaced by

procedure L; begin [declarations;] LG; L' end

and finally that

4) the governed statements of the decision and iteration statements as well as the last statements in compound statements and blocks be replaced simply by:

procedur L; S;

since in all these cases the successor is already specified by the containing statement. Knuth and Floyd conclude the exposition of their technique by stating "The program ends by calling a null procedure."

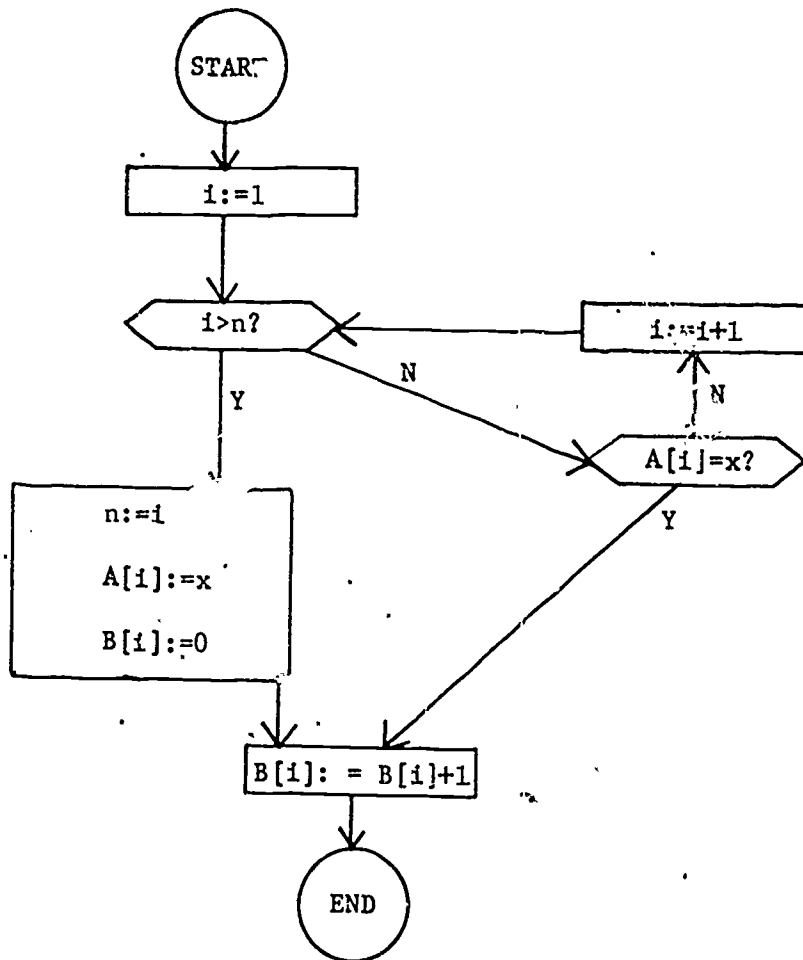
To see why this technique merely replaces the go to statement by an equivalent construct, namely the null procedure, we will apply the technique to a program already discussed by Knuth and Floyd. We will treat it as a "complete" program, omitting declarations, input, and output, in order to focus attention on the control flow. The example program, augmented with labels as required by the technique, is:

begin

L1: for i:=1 step 1 until n do

L2: if A[i]=x then L3: go to found;
 L4: not found: n:=i; L5: A[i]:=x; L6: B[i]=0;
 L7: found: B[i]:=B[i]+1;
end

The flow chart for this program is:



If we apply the stated rules to the example program the following procedures are obtained.

procedure L1; begin for i:=1 step 1 until n
 do L2; L4 end
procedure L2; if A[i]=x then L3;
procedure L3; L7;

procedure L4; begin n:=i; L5 end

procedure L5; begin A[i]:=x; L6 end

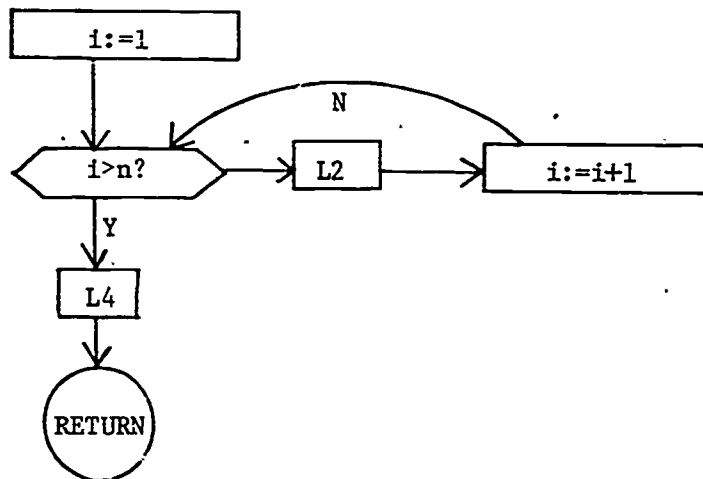
procedure L6; begin B[i]:=0; L7 end

procedure L7: begin B[i]:=B[i]+1; null end

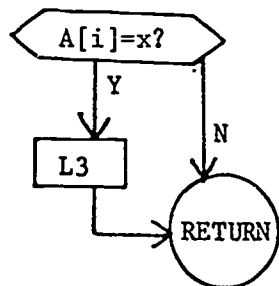
where we include the call on "null" in procedure L7 because statement L7 concludes the program.

The flowcharts of the above procedures are

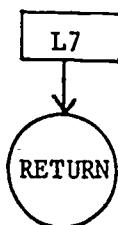
L1:

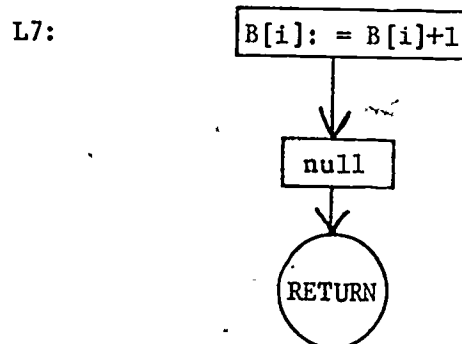
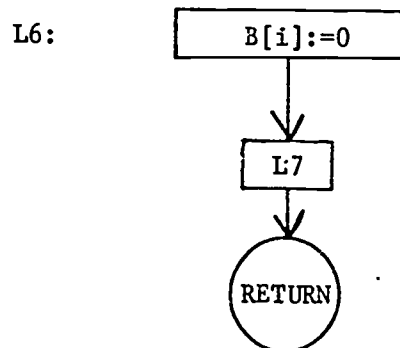
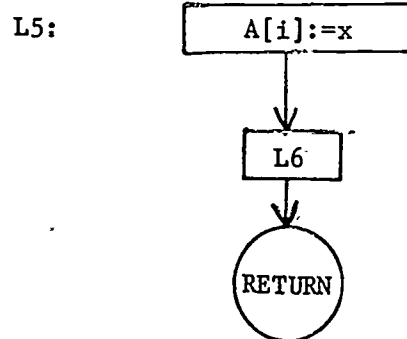
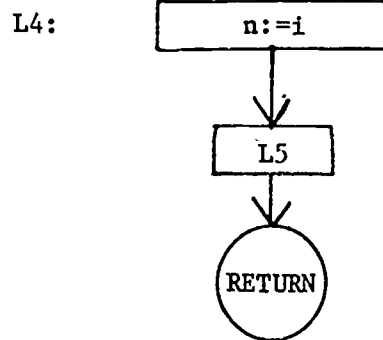


L2:

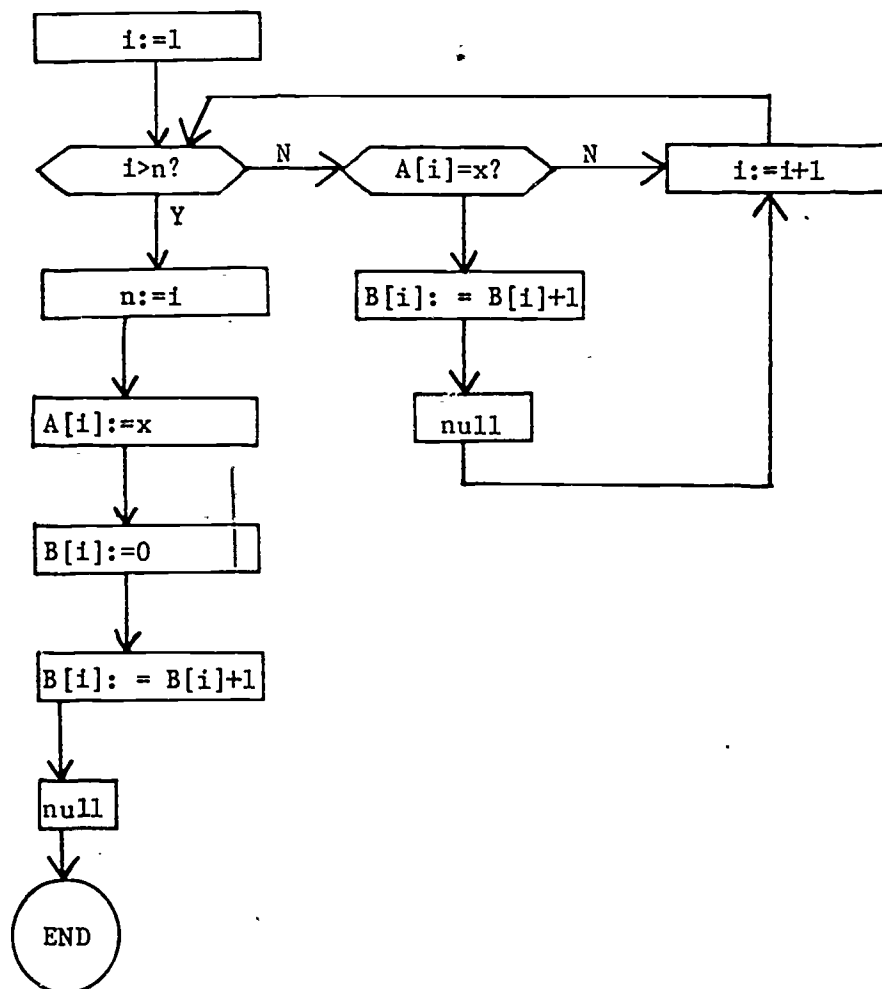


L3:





Substituting each at its point of invocation yields:



This flowchart is smooth, but does not represent the program because it does not show that the null procedure fails to return control to its point of invocation, but instead jumps to the end of the program. In ALGOL 60, a procedure can do this only by executing a go to statement.

Knuth and Floyd do give a smooth program using recursive procedure calls which is equivalent to their original non-smooth program:

procedure find;

if $i > n$ then begin $n:=i$; $A[i]:=x$; $B[i]:=0$

end;

else if $A[i] \neq x$ then

begin $i:=i+1$; find end;

$i:=1$; find; $B[i] := B[i]+1$;

They are able to do this only because of the simplicity of their original example. If, for example, we replace the statement L6 of their example by the semantically equivalent sequence:

L6: $B[i]:=1$; go to L8;

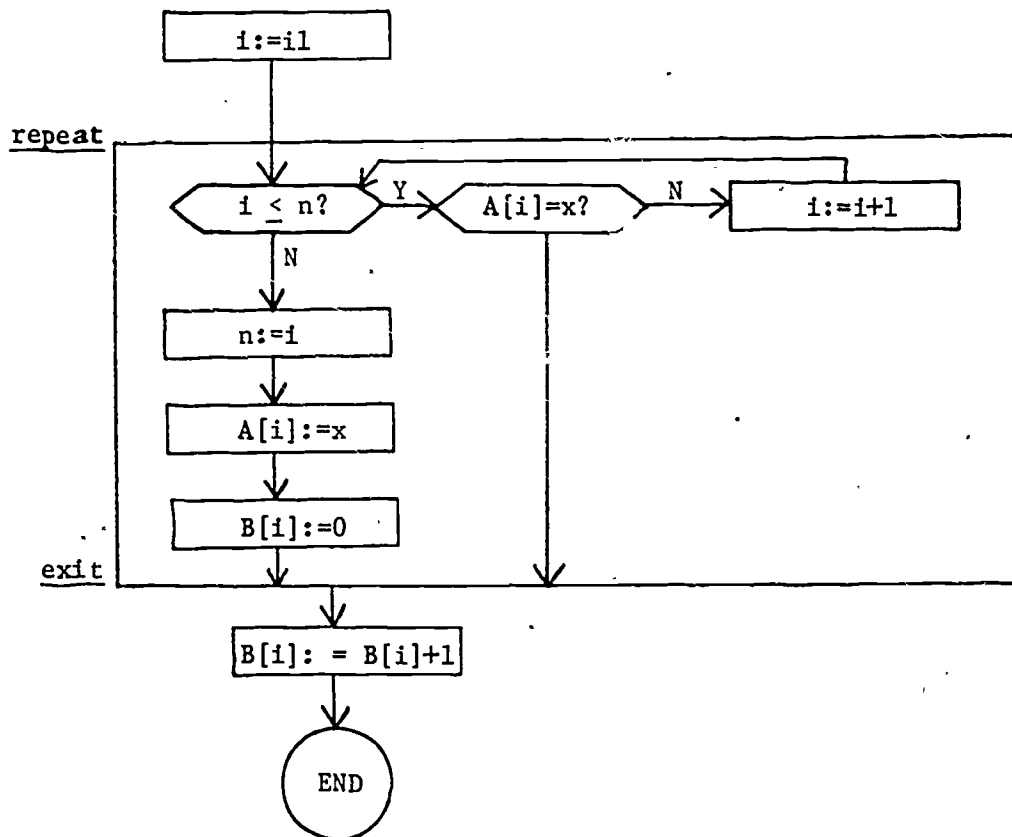
where L8 is a label placed on the end of the original program, their ad hoc transformation would no longer apply.

IV. The exit Statement

Knuth and Floyd suggest that a third way of writing their example program is with a special repeat statement and a restricted form of the go to called an exit statement. In this form, their example becomes:

```
begin i:=1;  
  repeat begin  
    while i ≤ n do if A[i]=x then exit else i:=i+1;  
    n:=i; A[i]:=x; B[i]:=0; exit  
  end;  
  B[i] := B[i]+1;  
end;
```

The flowchart of this example is:



This flowchart is clearly equivalent to the original.

Another restricted form of the go to is found in the go to-free language BLISS[9] where the construct: leave <label> with <expression> serves this purpose. This brings out the fact that the mere absence of the go to statement does not insure the smoothness of a programming language. On the other hand the absence of convenient block structuring features in FORTRAN would make the writing of even a smooth program in GO TO-less FORTRAN somewhat awkward.

V. Conclusions

In this paper we have shown that the recursive procedure method proposed by Knuth and Floyd does not, as they suggest, transform a non-smooth program into a smooth one. Instead it merely replaces the go to statement by a restricted form of the go to, namely the null procedure. We have seen that ALGOL 60 without the go to statement is a smooth language whether invocation of non-null procedures is allowed or not. Therefore we are left with Böhm and Jacopini's original result, namely that in general the transformation of non-smooth programs into smooth ones requires the introduction of additional variables or the equivalent recomputation of some quantities in the program.

It should be clear from the discussion that smoothness is not a property of the algorithm being implemented, but of the program which implements the algorithm. Whether it is desirable to retain, restrict, or eliminate the go to statement is a question of esthetics, since go to statements occurring in a program can always be replaced by additional code. It seems to us that a reasonable case has been made by Wulf for the replacement of the arbitrary go to by a restricted form, such as the leave construct of BLISS. Perhaps the term quasi-smooth could be used for non-smooth programs and languages using a restricted form of the go to.

It should be possible to find a use for the concept of the smooth program as a tool in the analysis of programs. One such application, to program analysis for parallel execution, has already been found and will be reported in a forthcoming paper. Other applications in the area of compiler optimization are being sought at the present time. Observations by other recent workers in this field, especially the remark by Lowry and Medlock [10] that most loops in FORTRAN programs are single entry loops, give us reason to

believe that the class of smooth programs includes a substantial subset of useful programs. We believe therefore that the class of smooth languages should also be of practical interest.

REFERENCES

1. Böhm, Corrado and Jacopini, Giuseppe. Flow diagrams, Turing machines, and languages with only two formation rules. Comm. ACM 9 (1966), 366-371.
2. Cooper, David C. Böhm and Jacopini's reduction of flowcharts. Comm. ACM 10 (1967), 463, 473.
3. Bruno, J. and Steiglitz, K. The expression of algorithms by charts. JACM 19 (1972), 517-525.
4. Dijkstra, E. Go to statement considered harmful. Comm. ACM 11 (1968), 147-148.
5. Knuth, D. E. and Floyd, R. W. Notes on avoiding "GO TO" statements. Information Processing Letters 1 (1971), 23-31.
6. Naur, Peter. (Ed.) Revised report on the algorithmic language ALGOL 60. Comm. ACM 6 (1963), 1-17.
7. Cocke, John. Global common subexpression elimination. SIGPLAN Notices 5, 7 (1970), 20-24.
8. Allen, Frances E. Control flow analysis. SIGPLAN Notices 5, 7 (1970), 1-19.
9. Wulf, William A. A case against the GO TO. Proc. ACM 72, 791-797.
10. Lowry, Edward S. and Medlock, C. Object code optimization. Comm. ACM 12 (1969), 13-22.